The Lifted Newton Method for Nonlinear Optimization

Moritz Diehl
Optimization in Engineering Center (OPTEC)
K.U. Leuven, Belgium
joint work with Jan Albersmeyer

Overview

- **Idea of Lifted Newton Method**
- An algorithmic trick to exploit structure "for free"
- Application to optimization
- Convergence analysis
- Numerical Tests

Lifted Newton Method in detail described in:

SIAM J. Optim.
Vol. 20, No. 3, pp. 1655–1684

THE LIFTED NEWTON METHOD AND ITS APPLICATION IN OPTIMIZATION*

JAN ALBERSMEYER† AND MORITZ DIEHL‡

Simplified Setting: Root Finding Problem

Aim: solve root finding problem

$$\boxed{F(u) = 0}$$

where $F \in \mathcal{C}^1(\mathbb{R}^{n_u}, \mathbb{R}^{n_u})$ is composed of several nonlinear subfunctions:

Simplified Setting: Root Finding Problem

Aim: solve root finding problem

$$\boxed{F(u) = 0}$$

where $F \in \mathcal{C}^1(\mathbb{R}^{n_u}, \mathbb{R}^{n_u})$ is composed of several nonlinear subfunctions:

Algorithm 1: Function with output of intermediate variables

Input : $u \in \mathbb{R}^{n_u}$
Output: $x_1 \in \mathbb{R}^{n_1}, \ldots, x_m \in \mathbb{R}^{n_m}, F \in \mathbb{R}^{n_u}$
begin
  for $i = 1, 2, \ldots, m$ **do**
    $\boxed{x_i = f_i(u, x_1, x_2, \ldots, x_{i-1});}$
  end for
  $\boxed{F := f_F(u, x_1, x_2, \ldots, x_m);}$
end

Idea: "Lift" root finding problem into a higher dimensional space…

Lifted Problem

The equivalent, "lifted" problem is:

$$\boxed{G(u,x) = 0}$$

with

$$G(u,x) = \begin{pmatrix} f_1(u) & - & x_1 \\ f_2(u,x_1) & - & x_2 \\ \vdots & & \\ f_m(u,x_1,\ldots,x_{m-1}) & - & x_m \\ f_F(u,x_1,\ldots,x_m) & & \end{pmatrix}$$

Why to lift and increase the system size?

- Lifting is generalization of "multiple shooting” for
  - boundary value problems [Osborne 1969]
  - ODE / PDE parameter estimation [Bock1987, Schloeder1988]
  - optimal control [Bock and Plitt 1984, Gill et al. 2000, Schaefer2005]

- Lifting offers advantages in terms of
  - sparsity exploitation (not today's focus)
  - more freedom for initialization
  - faster local contraction rate

Motivating Toy Example

- Original scalar root finding problem:

$$F(u) := u^{16} - 2 = 0$$

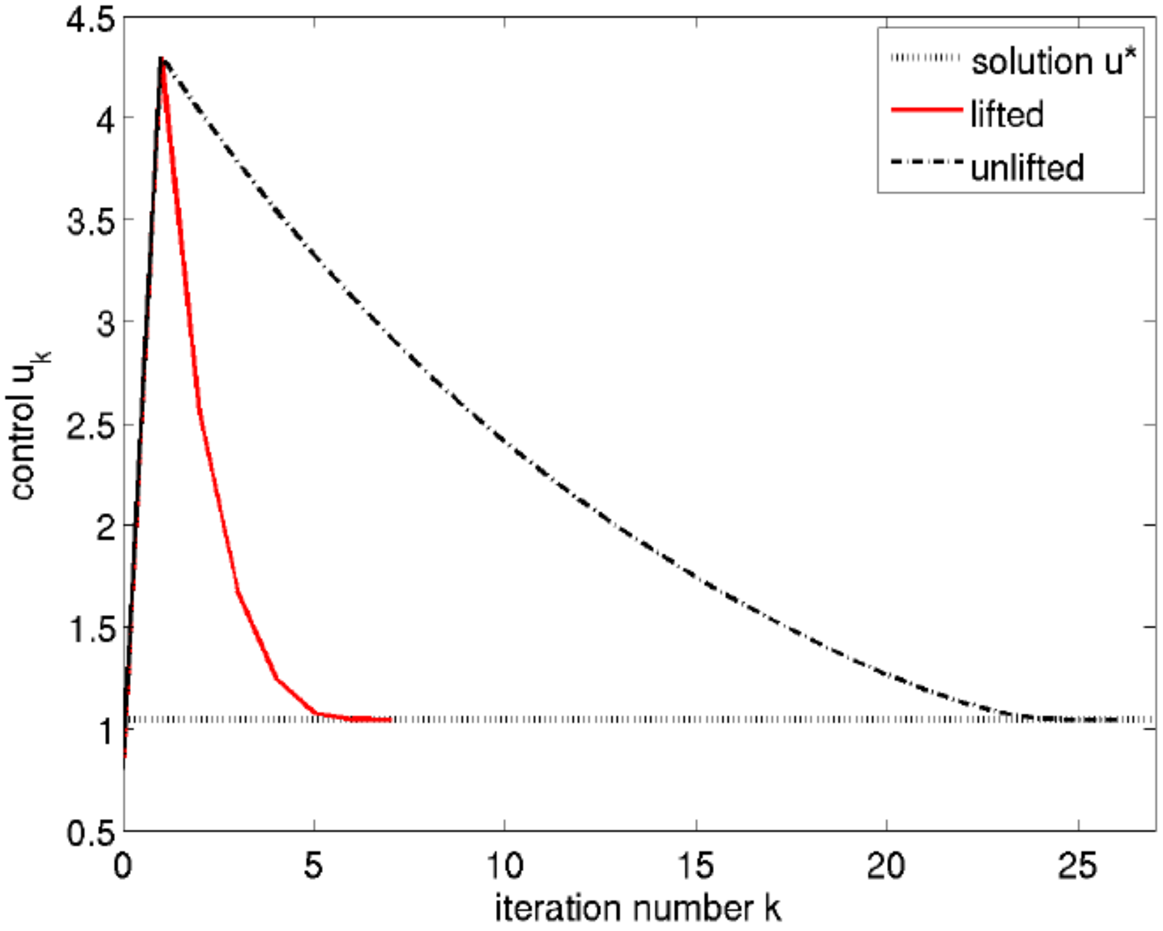
- Lifted formulation: 5 equations in 5 variables:

$$\begin{aligned} x_1 &:= u^2 & x_2 &:= x_1^2 \\ x_3 &:= x_2^2 & x_4 &:= x_3^2 \\ F &:= x_4 - 2 \end{aligned}$$

- Compare lifted and unlifted Newton iterates.
- Use same initial guess – obtain lifted variables by forward evaluation.

Motivating Toy Example

- First iteration is identical, as we initialized identically
- Lifted Newton method converges in 7 instead of 26 iterations!



Lifted Problem is much larger ... is it more expensive?

In each lifted Newton iteration

$$\begin{pmatrix} x^{k+1} \\ u^{k+1} \end{pmatrix} = \begin{pmatrix} x^{k} \\ u^{k} \end{pmatrix} + \begin{pmatrix} \Delta x^{k} \\ \Delta u^{k} \end{pmatrix}$$

we have to solve a large linear system:

$$\begin{pmatrix} \Delta x^{k} \\ \Delta u^{k} \end{pmatrix} = - \left[\frac{\partial G}{\partial (u,x)} (x^k, u^k) \right]^{-1} G(x^k, u^k)$$

It is large and structure can be exploited…

… but exploitation algorithms have so far been difficult to implement
[Schloeder 1988, Schaefer 2005].

Overview

- Idea of Lifted Newton Method
- **An algorithmic trick to exploit structure "for free"**
- Application to optimization
- Convergence analysis
- Numerical Tests

Algorithmic Trick: Preliminaries 1

Original "user function":

Algorithm 1: Function with output of intermediate variables

Input : $u \in \mathbb{R}^{n_u}$
Output: $x_1 \in \mathbb{R}^{n_1}, \ldots, x_m \in \mathbb{R}^{n_m}, F \in \mathbb{R}^{n_u}$
begin
  for $i = 1, 2, \ldots, m$ **do**
    $x_i = f_i(u, x_1, x_2, \ldots, x_{i-1});$
  end for
  $F := f_F(u, x_1, x_2, \ldots, x_m);$
end

"Lifted residual" (after minor code additions):

Algorithm 2: Residual function $G(u, y)$

Input : $u, y_1, \ldots, y_m$
Output: $G_1, \ldots, G_m, F$
begin
  for $i = 1, 2, \ldots, m$ **do**
    $x_i = f_i(u, x_1, x_2, \ldots, x_{i-1});$
    $\boxed{\begin{array}{l} G_i = x_i - y_i; \\ x_i = y_i; \end{array}}$
  end for
  $F = f_F(u, x_1, x_2, \ldots, x_m);$
end

Algorithmic Trick: Preliminaries 2

Write
$$G(u,x) = \begin{pmatrix} H(u,x) - x \\ f_F(u,x) \end{pmatrix}$$

with
$$H(u,x) := \begin{pmatrix} f_1(u) \\ f_2(u,x_1) \\ \vdots \\ f_m(u,x_1,\ldots,x_{m-1}) \end{pmatrix}$$

in each lifted Newton iteration, we have to solve:

$$H(u,x) - x + \left(\frac{\partial H}{\partial x}(u,x) - \mathbb{I}_{n_x}\right)\Delta x + \frac{\partial H}{\partial u}(u,x)\Delta u = 0$$
$$f_F(u,x) + \frac{\partial f_F}{\partial x}(u,x)\Delta x + \frac{\partial f_F}{\partial u}(u,x)\Delta u = 0$$

How to solve this linear system efficiently ?

Well-known condensing algorithm

Can eliminate

$$\Delta x = \underbrace{-\left(\frac{\partial H}{\partial x}(u,x) - \mathbb{I}_{n_x}\right)^{-1} (H(u,x) - x) +}_{=:a} \underbrace{-\left(\frac{\partial H}{\partial x}(u,x) - \mathbb{I}_{n_x}\right)^{-1} \frac{\partial H}{\partial u}(u,x)\, \Delta u}_{=:A}$$

to get "condensed" system in unlifted dimensions:

$$\begin{array}{ccccc} 0 & = & f_F(u,x) + \frac{\partial f_F}{\partial x}(u,x)a & + & \left(\frac{\partial f_F}{\partial u}(u,x) + \frac{\partial f_F}{\partial x}(u,x)A\right) & \Delta u \\ & =: & b & + & B & \Delta u. \end{array}$$

First solve this sytem: $\Delta u = -B^{-1}\, b$

Then expand solution: $\Delta x = a + A\, \Delta u$

But how to compute a, b, A, B efficiently ?

Basis of new trick: an auxiliary function

Define $Z(u,d)$

as implicit function satisfying a perturbed fixed point equation (by vector d)

$$H(u,z) - z - d = 0$$

PROPOSITION: if $d = H(u,x) - x$ then

$$\frac{\partial Z}{\partial u}(u,d) = -\left(\frac{\partial H}{\partial x}(u,x) - \mathbb{I}_{n_x}\right)^{-1} \frac{\partial H}{\partial u}(u,x)$$

and

$$\frac{\partial Z}{\partial d}(u,d) = \left(\frac{\partial H}{\partial x}(u,x) - \mathbb{I}_{n_x}\right)^{-1}$$

But how to obtain Z ?

Obtain Z by another minor code modification

- Can show that Z is obtained by one evaluation of the following function:

Algorithm 3: Modified function $Z(u, d)$

Input : $u, d_1, \ldots, d_m$
Output: $z_1, \ldots, z_m, F$
begin
    for $i = 1, 2, \ldots, m$ **do**
        $x_i = f_i(u, x_1, x_2, \ldots, x_{i-1});$
        $\boxed{z_i = x_i - d_i;}$
        $\boxed{x_i = z_i;}$
    end for
    $F = f_F(u, x_1, x_2, \ldots, x_m);$
end

Costs of generating and solving linear system

Using Z, can easily compute a, A, b, B via directional derivatives:

$$a = -\frac{\partial Z}{\partial d}(u,d)\; d \qquad A = \frac{\partial Z}{\partial u}(u,d)$$

$$b = \quad f_F(u,x) + \tfrac{\partial f_F}{\partial x}(u,x)a \qquad B = \tfrac{\partial f_F}{\partial u}(u,x) + \tfrac{\partial f_F}{\partial x}(u,x)A$$

and then compute the Newton step

$$\Delta u = -B^{-1}\; b$$

$$\Delta x = a + A\;\Delta u$$

Computational effort per iteration:

- computing A & B (in one combined forward sweep)
- factoring B

Small extra efforts for lifting [cf. Schloeder 1988]:

- computing vector a (one extra directional derivative)
- matrix vector product $A\;\Delta u$

Overview

- Idea of Lifted Newton Method
- An algorithmic trick to exploit structure "for free"
- **Application to optimization**
- Convergence analysis
- Numerical Tests

Two lifted algorithms

- Lifted Gauss-Newton Method
- Lifted SQP Method

Lifted Gauss-Newton

- Original Problem:

$$\begin{aligned} \min_{u} \quad & \frac{1}{2}\|F_1(u)\|_2^2 \\ \text{s.t.} \quad & \\ & F_2(u) \begin{Bmatrix} = \\ \geq \end{Bmatrix} 0 \end{aligned}$$

- Lifted Problem:

$$\begin{aligned} \min_{u,x} \frac{1}{2}\|f_{F_1}(u,x)\|_2^2 & \\ \text{s.t.} \quad & \\ f_{F_2}(u,x) \begin{Bmatrix} = \\ \geq \end{Bmatrix} & \; 0 \\ H(u,x) - x \;\; = & \;\; 0 \end{aligned}$$

(obtained by lifting $F(u) := (F_1(u)^T, F_2(u)^T)^T$)

Lifted Gauss-Newton: Quadratic Subproblems

- Linearized lifted problem = structured QP subproblem:

$$\begin{array}{rl}
\min\limits_{\Delta u, \Delta x} & \frac{1}{2}\left\| f_{F_1}(u_k, x_k) + \frac{\partial f_{F_1}}{\partial(u,x)}(u_k, x_k)\begin{pmatrix}\Delta u\\ \Delta x\end{pmatrix}\right\|_2^2 \\
\text{s.t.} & \\
 & f_{F_2}(u_k) + \frac{\partial f_{F_2}}{\partial(u,x)}(u_k, x_k)\begin{pmatrix}\Delta u\\ \Delta x\end{pmatrix} \left\{\begin{matrix} = \\ \geq \end{matrix}\right\} 0 \\
H(u_k, x_k) - x_k + \frac{\partial H}{\partial(u,x)}(u_k, x_k)\begin{pmatrix}\Delta u\\ \Delta x\end{pmatrix} - \Delta x & = \quad 0
\end{array}$$

Lifted Gauss-Newton: Quadratic Subproblems

- Linearized lifted problem = structured QP subproblem:

$$
\begin{aligned}
\min_{\Delta u, \Delta x} \quad & \frac{1}{2} \left\| f_{F_1}(u_k, x_k) + \frac{\partial f_{F_1}}{\partial(u,x)}(u_k, x_k) \begin{pmatrix} \Delta u \\ \Delta x \end{pmatrix} \right\|_2^2 \\
\text{s.t.} \quad & \\
& f_{F_2}(u_k) + \frac{\partial f_{F_2}}{\partial(u,x)}(u_k, x_k) \begin{pmatrix} \Delta u \\ \Delta x \end{pmatrix} \begin{Bmatrix} = \\ \geq \end{Bmatrix} 0 \\
& H(u_k, x_k) - x_k + \frac{\partial H}{\partial(u,x)}(u_k, x_k) \begin{pmatrix} \Delta u \\ \Delta x \end{pmatrix} - \Delta x = 0
\end{aligned}
$$

- Condensed QP (solved by dense QP solver):

$$
\begin{aligned}
\min_{\Delta u} \quad & \frac{1}{2} \|b_1 + B_1 \Delta u\|_2^2 \\
\text{s.t.} \quad & \\
& b_2 + B_2 \Delta u \begin{Bmatrix} = \\ \geq \end{Bmatrix} 0.
\end{aligned}
$$

easily get $b = (b_1^T, b_2^T)^T$ and $B = (B_1^T, B_2^T)^T$ as lifted derivatives of $F(u) := (F_1(u)^T, F_2(u)^T)^T$

Two lifted algorithms

- Lifted Gauss-Newton Method
- **Lifted SQP Method**

Lifted SQP Method: Problem Statement

- Regard unconstrained optimization problem: $\min\limits_{u} \; \varphi(u)$
- Lifted formulation:

$$
\begin{array}{ll}
\min\limits_{u,w} & f_{\varphi}(u, w_1, w_2, \ldots, w_m) \\
\text{s.t.} & \\
& g(u,w) = \begin{pmatrix} f_1(u) & - & w_1 \\ f_2(u, w_1) & - & w_2 \\ \vdots & & \\ f_m(u, w_1, \ldots, w_{m-1}) & - & w_m \end{pmatrix} = 0
\end{array}
$$

- Aim is to automatically get efficient implementation of large SQP method
- Idea: Lift root finding problem:

$$F(u) := \nabla_u \varphi(u) \; = 0$$

- Question: Which variables shall be lifted ?

Gradient evaluation via adjoint differentiation

To compute $\boxed{F(u) := \nabla_u \varphi(u)}$ perform the following code:

$$w_1 = f_1(u) \tag{3.8a}$$

$$w_2 = f_2(u, w_1) \tag{3.8b}$$

$$\vdots$$

function value

$$w_m = f_m(u, w_1, \ldots, w_n - 1) \tag{3.8c}$$

$$y \equiv f_\varphi(u, w_1, \ldots, w_m) \tag{3.8d}$$

$$\bar{w}_m = \nabla_{w_m} f_\varphi \tag{3.8e}$$

$$\bar{w}_{m-1} = \nabla_{w_{m-1}} f_\varphi + \nabla_{w_{m-1}} f_m \bar{w}_m \tag{3.8f}$$

$$\vdots$$

$$\bar{w}_1 = \nabla_{w_1} f_\varphi + \sum_{i=2}^{m} \nabla_{w_1} f_i \bar{w}_i \tag{3.8g}$$

gradient

$$\bar{u} = \nabla_u f_\varphi + \sum_{i=1}^{m} \nabla_u f_i \bar{w}_i . \tag{3.8h}$$

Gradient evaluation via adjoint differentiation

To compute $\boxed{F(u) := \nabla_u \varphi(u)}$ perform the following code:

$$w_1 = f_1(u) \tag{3.8a}$$

$$w_2 = f_2(u, w_1) \tag{3.8b}$$

$$\vdots$$

function value

$$w_m = f_m(u, w_1, \dots, w_n - 1) \tag{3.8c}$$

$$y \equiv f_\varphi(u, w_1, \dots, w_m) \tag{3.8d}$$

$$\bar{w}_m = \nabla_{w_m} f_\varphi \tag{3.8e}$$

$$\bar{w}_{m-1} = \nabla_{w_{m-1}} f_\varphi + \nabla_{w_{m-1}} f_m \bar{w}_m \tag{3.8f}$$

$$\vdots$$

$$\bar{w}_1 = \nabla_{w_1} f_\varphi + \sum_{i=2}^{m} \nabla_{w_1} f_i \bar{w}_i \tag{3.8g}$$

gradient

$$\bar{u} = \nabla_u f_\varphi + \sum_{i=1}^{m} \nabla_u f_i \bar{w}_i. \tag{3.8h}$$

Lifted Newton equivalent to Full Space SQP

THEOREM: If we lift $F(u) := \nabla_u \varphi(u)$ with respect to all intermediate variables: $x = (w_1, \ldots, w_m, \bar{w}_m, \ldots, \bar{w}_1)$

then full space SQP and lifted Newton iterations are identical, with $\lambda \equiv \bar{w}$

COROLLARY: Same equivalence holds for lifting of constrained problems:

$$\boxed{\begin{aligned} &\min_u \varphi(u) \\ &\text{subject to} \\ &\qquad h(u) = 0, \end{aligned}} \longrightarrow \boxed{\begin{aligned} &\min_{u,w} f_\varphi(u,w) \\ &\text{subject to} \\ &\qquad g(u,w) = 0, \\ &\qquad f_h(u,w) = 0, \end{aligned}}$$

if we lift Lagrange gradient and constraint:

$$\begin{pmatrix} \nabla_u \mathcal{L}^{\text{orig}} \\ h \end{pmatrix}$$

Overview

- Idea of Lifted Newton Method
- An algorithmic trick to exploit structure "for free"
- Application to optimization
- **Convergence analysis**
- Numerical Tests

Why does lifting often improve convergence speed ?

Simple model problem: chain of scalar functions

- Regard a sequence of scalar functions:

$$x_1 = f_1(u),\ x_2 = f_2(x_1), \ldots, x_m = f_m(x_{m-1}), \text{and } f_F(x_m) \equiv f_{m+1}(x_m)$$

- after affine transformations, can assume that solution is zero and

$$f_i(x) = x + b_i(x)^2 + O(|x|^3)$$

- under these circumstances, the non-lifted function is:

$$F(u) = f_{m+1}(f_m(\ldots f_1(u)\ldots)) = u + \Big(\sum_{i=1}^{m+1} b_i\Big)u^2 + O(|u|^3)$$

Convergence speed of Non-Lifted Newton

- Derivative is given by

$$F'(u) = 1 + 2\Big(\sum_{i=1}^{m+1} b_i\Big)u + O(|u|^2)$$

- It is easy to show that non-lifted Newton iterations contract like:

$$u^{[k+1]} = \Big(\sum_{i=1}^{m+1} b_i\Big)(u^{[k]})^2 + O(|u^{[k]}|^3)$$

i.e. quadratic convergence with contraction constant $\left(\sum_{i=1}^{m+1} b_i\right)$

Lifted Newton Convergence

- Lifted residual is

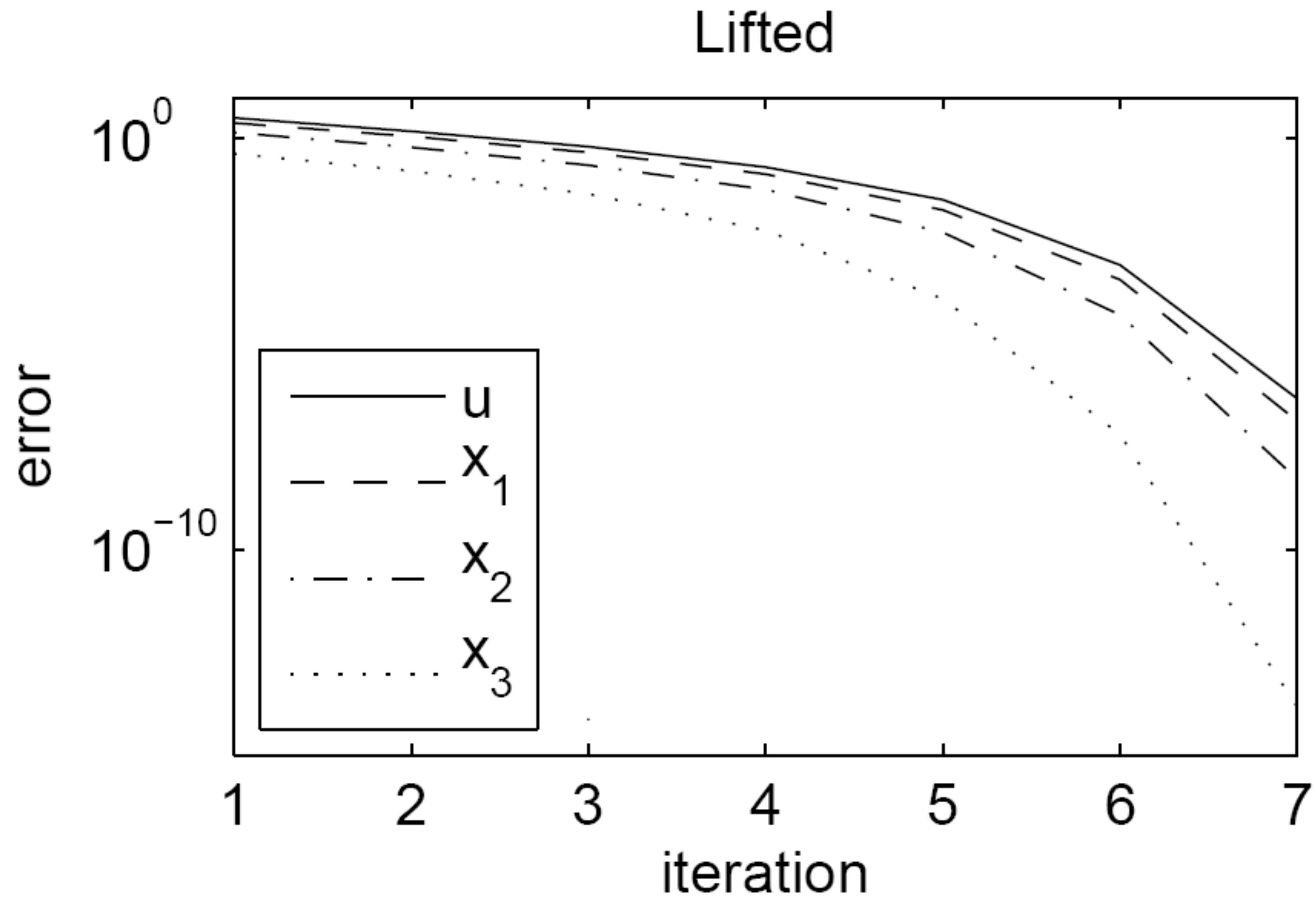
$$G(u,x) = \begin{pmatrix} u + b_1 u^2 & - & x_1 \\ x_1 + b_2 x_1^2 & - & x_2 \\ \vdots & & \\ x_{m-1} + b_m x_{m-1}^2 & - & x_m \\ x_m + b_{m+1} x_m^2 & & \end{pmatrix} + O\left(\left\| \begin{pmatrix} u \\ x \end{pmatrix} \right\|^3 \right)$$

THEOREM: Lifted Newton iterations contract in a "staggered" way:

$$\begin{pmatrix} u \\ x_1 \\ \vdots \\ x_{m-1} \\ x_m \end{pmatrix}^{[k+1]} = \begin{pmatrix} b_1 \left(u^{[k]}\right)^2 + \sum_{i=2}^{m+1} b_i \left(x_{i-1}^{[k]}\right)^2 \\ \sum_{i=2}^{m+1} b_i \left(x_{i-1}^{[k]}\right)^2 \\ \vdots \\ b_m \left(x_{m-1}^{[k]}\right)^2 + b_{m+1} \left(x_m^{[k]}\right)^2 \\ b_{m+1} \left(x_m^{[k]}\right)^2 \end{pmatrix} + O\left(\left\| \begin{pmatrix} u \\ x \end{pmatrix}^{[k]} \right\|^3 \right)$$

(if all b_i have the same sign, last variable is "leader" and contracts fastest)

Convergence for motivating toy example



Practical Conclusions from Theorem

Two cases:

- Same curvature → lifted Newton better.
  E.g. in simulation with iterative calls of same time stepping function

- Opposite curvature → unlifted Newton better.
  e.g. $F(u) = u$ decomposed as $f_1(u) = u^2 \quad f_2(x) = \sqrt{x}$
  Unlifted Newton converges in first iteration, lifted not!

Overview

- Idea of Lifted Newton Method
- An algorithmic trick to exploit structure "for free"
- Application to optimization
- Convergence analysis
- **Numerical Tests**

Large Example: Shallow Water Equation [Copsey 09]

- Regard 2-D shallow water wave equation (with unknown parameters)

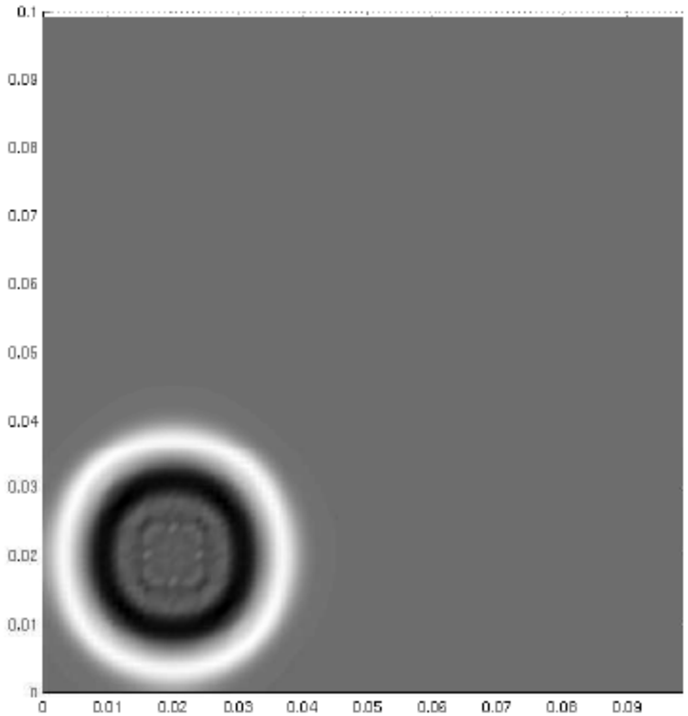
$$
\begin{aligned}
\partial_t u(t,x,y) &= -g\partial_x h(t,x,y) - \boxed{b} u(t,x,y) \\
\partial_t v(t,x,y) &= -g\partial_y h(t,x,y) - \boxed{b} v(t,x,y) \\
\partial_t h(t,x,y) &= -\boxed{H}\left[\partial_x u(t,x,y) + \partial_y v(t,x,y)\right]
\end{aligned}
$$

- Discretize all 3 states u, v, h on 30 x 30 grid and perform 10000 timesteps (= 27 million variables !)

- Measure **only height** h every 100 time steps (= 90 000 measurements)

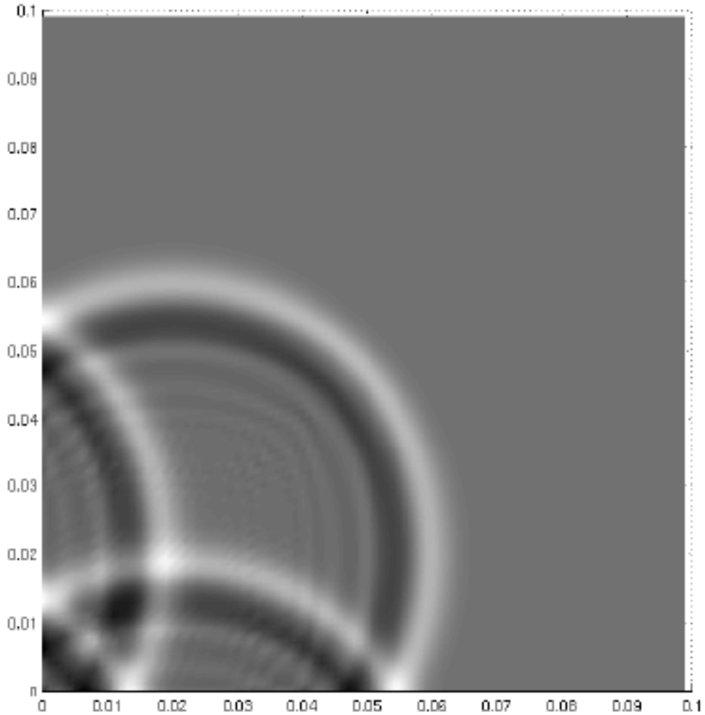
- Aim: estimate 2 unknown parameters, water depth H and viscous friction coefficient b

Shallow Water Example: Measurements



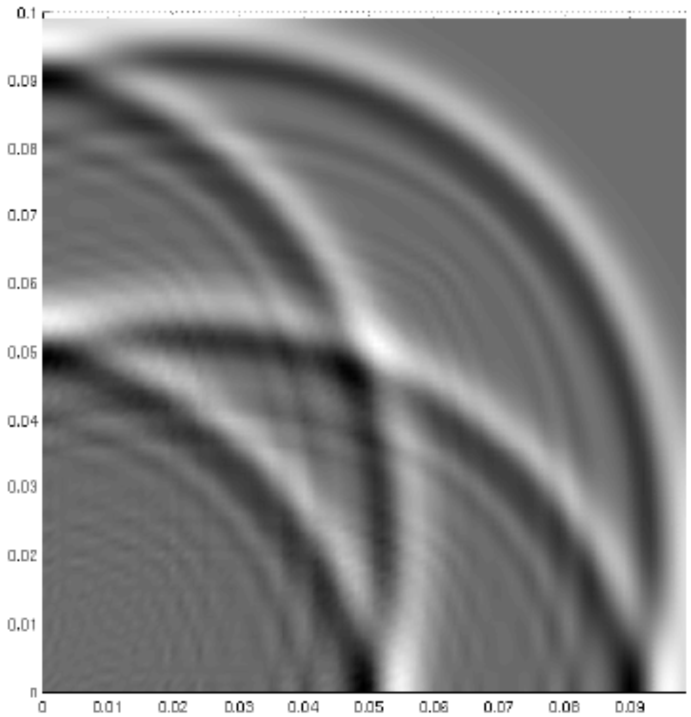
(a)

Shallow Water Example: Measurements



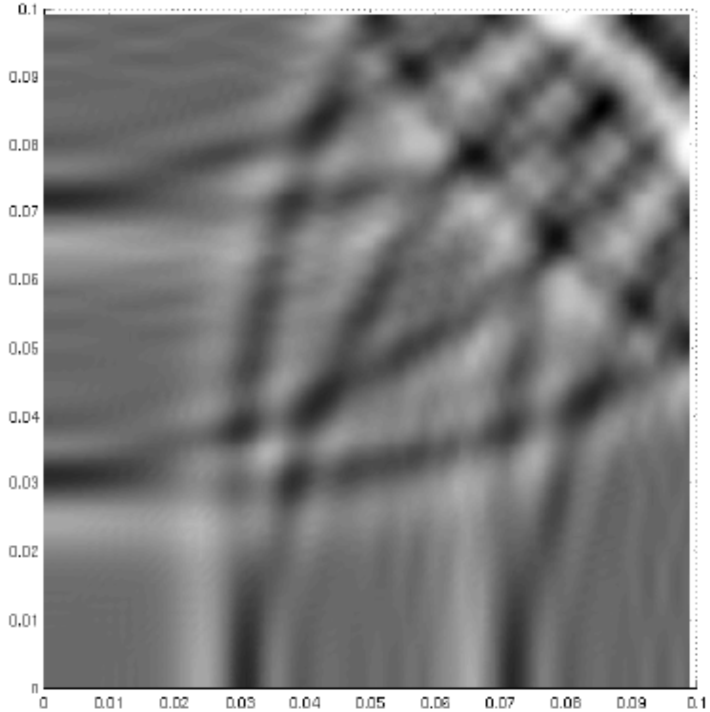
(b)

Shallow Water Example: Measurements



(c)

Shallow Water Example: Measurements



(d)

Shallow Water: Compare three Gauss-Newton variants

(A) Non-lifted Newton with 2 variables only

(B) Lifted Newton (with 90 000 lifted variables), initialized "automatically", like non-lifted variant via a forward simulation

(C) Lifted Newton, but use all 90 000 height measurements for initialization

Iteration count for three methods (diff. initial guess)

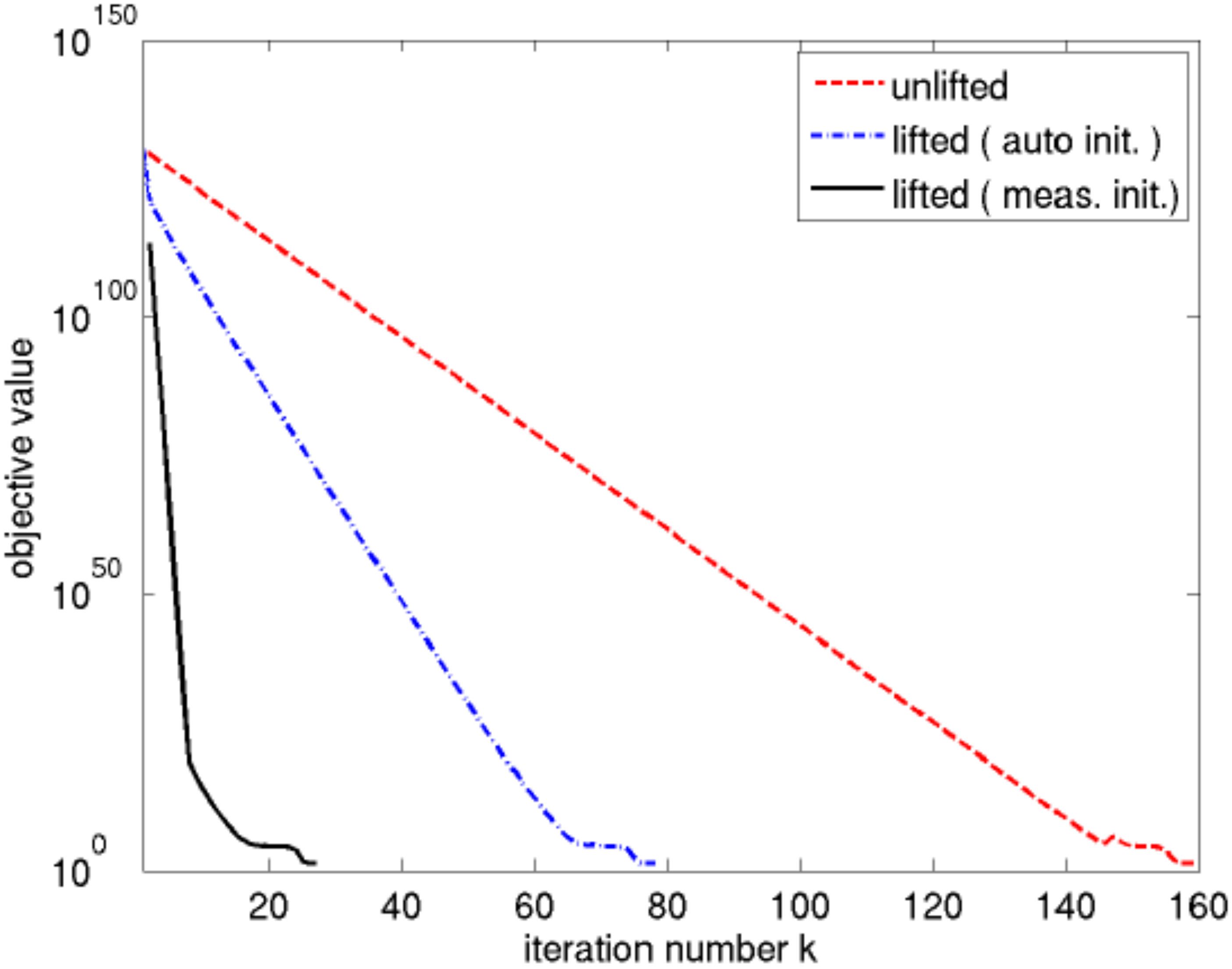
b	H	(A) # iterations unlifted	(B) #iterations lifted (autom. init.)	(C) #iterations lifted (meas init.)
0.5	0.01	5	5	4
5	0.01	6	5	4
15	0.01	17	7	6
30	0.01	27	7	6
2	0.005	31	9	5
2	0.02	38	12	5
2	0.1	44	13	8
0.2	0.001	33	12	7
1	0.005	47	10	5
4	0.02	$\boxed{56}$	$\boxed{10}$	$\boxed{5}$
1	0.02	44	9	6
20	0.001	24	10	6

true values $b = 2$ and $H = 0.01$

CPU time per iteration: 9 s for unlifted, 12 s for lifted

(Note: formulation e.g. in AMPL would involve 27 million variables)

Convergence for another PDE parameter estimation example



Summary

- Lifting offers advantages for Newton type optimization:
  - faster local convergence rate (observed & proven in simplified setting)
  - more freedom for initialization

- Structure exploiting "Lifted Newton Methods" can easily be generated for any given user functions and any Newton type method:
  - only minor code additions
  - nearly no additional costs per iteration
  - compatible with SQP, BFGS, Gauss-Newton, …
  - compatible with any linear solver for condensed systems